AFIT/EN-TR-93-2

Air Force Institute of Technology

Ada/X Window System Bindings: Conversion Strategies

Karl S. Mathias
Capt, USAF

Mark A. Roth
Maj, USAF

18 March 1993

Ada/X Window System Bindings: Conversion Strategies

Karl S. Mathias
Mark A. Roth[1]

Abstract

The X Window System has come to be accepted as the standard for developing graphical user interfaces on mid to high range workstations. Ada interfaces to X through the use of bindings were developed in the late 1980's under the Software Technology for Adaptable Reliable Systems (STARS) contracts. The bindings have not been maintained, and as they become more obsolete, many are converting their applications to newer sets of commercially available bindings. This paper discusses how the bindings work and two strategies for converting out of the STARS bindings.

1 Introduction

The X Window System was developed by the Massachusetts Institute of Technology (MIT) in partnership with Digital Equipment Corporation (DEC). Released in 1986, it supplies a flexible, object-oriented, graphic toolkit that can be used to develop user interfaces. Now in the fifth release of its second major version (X11R5), X has become the de-facto standard interface with workstation manufacturers [5].

Ada bindings to X were developed in the late 1980's under the Software Technology for Adaptable Reliable Systems (STARS) contracts. The bindings do not cover the full X specification, have various execution problems, and are not maintained [3]. As new versions of the X Window System are released, the STARS bindings become increasingly obsolete.

This paper discusses strategies for converting from these old bindings to the newer bindings that are being commercially supported. An overview of the X Window System is given along with a discussion of how Ada binds to X. Examples are presented that demonstrate how the strategies for conversion have been used to convert Saber (an air/land battle wargame) from STARS to bindings developed by Software Engineering Research Corporation (SERC).

2 The X Window System: A Brief Overview

The X Window System uses a client-server architecture that is designed to take advantage of networked systems. A server process runs on each display machine in a network (see Figure 1); it functions as the interface to that device's hardware. Client applications communicate with the server to have it display images and react when the user takes specific actions. Clients and servers may run on the same or different machines.

An X client application may utilize several layers of software, as shown in Figure 2. Each layer is an independent set of library calls, with each higher level having complete access to any of the lower layers.

[1]The authors are with the Department of Electrical and Computer Engineering (AFIT/ENG), Air Force Institute of Technology, 2950 P ST, Wright-Patterson AFB, OH 45433-7765.

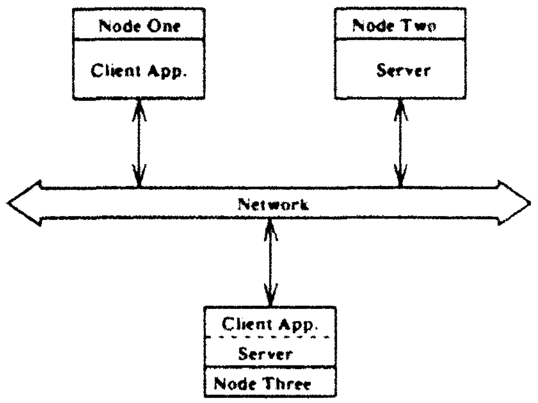

Figure 1: Client-Server Architecture of the X Window System

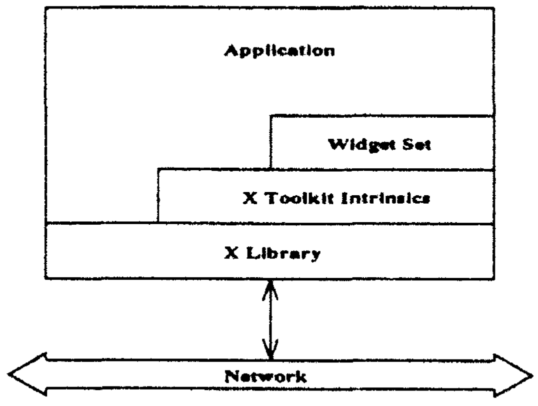

Figure 2: Architecture of X Window System Applications

2.1 X Library

The lowest-level functionality of the X Window System is the X Library (Xlib). This library provides the very basic functions needed to render images on a display. Examples would be drawing lines, defining display colors, handling keystrokes, etc.

2.2 X Toolkit Intrinsics

Programming in Xlib can be a very labor intensive task due to the amount of event handling that takes place. Various toolkits have been developed that alleviate the burden of programming in Xlib, the most popular being the X Toolkit Intrinsics (Xt). Applications using Xt are given general facilities that allow streamlined event handling and the ability to create reusable display objects (like windows or buttons) called *widgets*.

2.3 Widget Sets

The charter of X is "mechanism without policy" [6, 8]. Thus, unlike some windowing systems, X does not force developers to have a mandated style to their application. Commercial widget sets such as the Open Software Foundation's (OSF) Motif widget set fill in this gap. Layered on top of Xt, this library is used according to a style guide, and provides a consistent look and feel across applications written with it [2].

3 Ada Bindings to X

The X Window System is written in C without consideration for language independence. This is a problem for programmers using Ada to develop projects. Language bindings to C libraries offer a simple solution for programs needing X functionality.

3.1 Binding to C

In general, a binding to C will consist of four parts: declarations, preprocessing, C call(s), and post-processing. Figure 3 illustrates a typical binding. This binding takes one parameter as

```
1 : function Ada_Call(Param_One: in Special) return Special is
2 :   function C_Call(X: in INTEGER) return INTEGER;
3 :   pragma INTERFACE(C, C_Call);
4 :   pragma INTERFACE_NAME(C_Call, "_C_Call");
5 :
6 :   New_Param_One : INTEGER;
7 :   Return_Value  : INTEGER;
8 :
9 : begin
10:   New_Param := Special_To_INTEGER(Param_One);
11:   Return_Value := C_Call(New_Param);
12:   return(INTEGER_To_Special(Return_Value));
13: end Ada_Call;
```

Figure 3: An Example Ada-to-C Binding

input, converts it to an integer, and passes that integer to the corresponding C procedure.

Lines 1–4 declare the Ada binding function and the corresponding C function. The INTERFACE statements let the compiler and linker know that a C calling protocol will be needed, and the name of the function in the C libraries. Line 10 illustrates a simple preprocessing segment of code. The parameter Param_One must be converted to an integer value before the C call can accept it. An externally defined function accomplishes this. Line 11 calls the actual C function and passes it the converted value of Param_One. Line 12 post-processes the returned integer into the desired type. This is then returned to the calling routine.

3.2 Binding Thickness

Figure 3 is a simple mapping of a C function to an Ada call. A more complex C call, however, could require much more massaging in the pre- and post-processing areas to send and return values properly. The level to which a binding processes information before passing it on has come to be described in terms of *thickness*.

A thin binding set is characterized by a one-to-one mapping of Ada and C functions. Ada specific features are only used when a C-specific construct may not be available. Within the bindings, minimal amounts of processing are applied before or after the call to C.

Thick bindings are characterized by heavy use of Ada constructs and language features, generalization of the mapping to C calls (one-to-many mappings), and the addition of utility procedures/functions. A thick binding seeks to insulate the Ada programmer from the C calls while giving them the freedom to utilize Ada features to their best advantage.

These become important considerations when moving between binding sets. Moving from a thick to a thin binding will involve the creation of calls that were not previously required. Moving from a thin to thick binding may call for a redesign of some code sections to make optimum use of Ada's features.

3.3 STARS Binding Set

Three major sets of X interfaces are distributed by the STARS Foundation: Science Applications International Corp (SAIC) Xlib bindings, Boeing's Xt Intrinsics/OSF Motif bindings, and the Unisys Ada/Xt software. The Unisys software is an Ada implementation—the Xlib and Xt Intrinsics libraries have been re-coded in Ada.

The SAIC bindings cover most of the Xlib functions, while the Boeing bindings cover some Xlib, a large part of Xt, and most of the OSF Motif functions. While carrying the characteristics of thin bindings, they have some utility functions and Ada-specific constructs. As they were one of

the first sets of bindings available, many applications used them as an interface to X.

There are problems with using these bindings. There is little documentation available, so the programmer usually must read the source code to determine usage. The bindings have bugs, and users of the bindings must correct them [1]. The binding set is not complete and some bindings must be created. The SAIC and Boeing bindings are not entirely compatible, and use of one can mean loss of some functionality in the other [3]. Finally, there is no on-going support for the bindings and they are rapidly becoming obsolete.

3.4 Commercial Binding Sets

The lack of support for STARS has resulted in the emergence of commercially supported bindings. Bindings such as Ada/Motif by SERC are based upon the STARS bindings, but have been debugged, expanded and enhanced to make them a viable product [7].

The availability of these bindings makes it possible for applications to be upgraded to the latest releases of the X Window System and OSF Motif.

4 Converting Bindings

With many of the commercial bindings available based on the STARS bindings, converting a STARS application to a newer set would seem a simple task. Small applications do convert easily. Large applications, however, require a careful approach or excessive work will be done for a product that is less than optimal. This section discusses two strategies for conversion: low-level and high-level mapping.

4.1 Low-Level Mapping Strategy

Using the low-level style of conversion, the programmer determines either all at once, or incrementally all the STARS functions, procedures, and types in an application. Each of these is then iteratively converted to a corresponding function/procedure/type in the target binding set.

4.1.1 Method

Previous work by Moore [4] describes a straightforward method for converting each STARS identifier:

1. *Find Base STARS Declaration.* Identify the STARS source code statements that declared the identifier. This process may need to be nested, as the goal is to locate a standard Ada type definition.

2. *Find Similar Identifier.* Search the target binding specification for an identifier with the same or similar name as the STARS identifier.

3. *Find Base Target Binding Declaration.* Identify in the target binding how this identifier is declared. This process may also need to be repeated to locate the base Ada type declaration.

4. *Convert Between Base Declarations.* Using the declarations from above, the base type of the STARS identifier is converted to an appropriate type used by the target binding identifier.

4.1.2 Example Conversion

Saber, an air/land battle wargame, was originally coded using the STARS bindings. Figure 4 shows a piece of the code used to set the colors of terrain features. In this example, the code will be converted to the Ada/Motif binding set using low-level mapping.

1. *Find Base STARS Declarations.* The only STARS identifier used in Figure 4 is `XT.Pixel` (a type identifier). Looking in the STARS declarations `XT.Pixel` is defined as a subtype of `AFS_LARGE_NATURAL`. Looking in the boeing_afs package, `AFS_LARGE_NATURAL` is a subtype of `AFS_LARGE_INTEGER`. Finally, it is found that `AFS_LARGE_INTEGER` is defined as an Ada INTEGER. This is the base Ada type that is needed.

```
procedure Initialize_Terrain_Colors( Political_Red     : XT.Pixel;
                                     Political_Blue    : XT.Pixel;
                                     Political_Neutral : XT.Pixel;
                                     .
                                     .
                                     .
                                     City              : XT.Pixel ) is

  begin
    Political_Red_Color     := Political_Red;
    Political_Blue_Color    := Political_Blue;
    Political_Neutral_Color := Political_Neutral;
    .
    .
    .
    City_Color              := City;
  end Initialize_Terrain_Colors;
```

Figure 4: Section of Saber's STARS code

```
procedure Initialize_Terrain_Colors( Political_Red     : X_Lib.Pixel;
                                     Political_Blue    : X_Lib.Pixel;
                                     Political_Neutral : X_Lib.Pixel;
                                     .
                                     .
                                     .
                                     City              : X_Lib.Pixel ) is

  begin
    Political_Red_Color     := Political_Red;
    Political_Blue_Color    := Political_Blue;
    Political_Neutral_Color := Political_Neutral;
    .
    .
    .
    City_Color              := City;
  end Initialize_Terrain_Colors;
```

Figure 5: Low-level conversion of code

2. *Find Similar Identifier.* There is no `Pixel` in Ada/Motif's Xt Intrinsics library package, however there is one in the X library package.

3. *Find Base Target Binding Declaration.* Following a procedure similar to that above, `X_Lib.Pixel` will be have a base type of an integer range `(-2 ** 31 .. (2 ** 31) - 1)`.

4. *Convert Between Base Declarations.* Since both `Pixels` are based upon integer values, the conversion is simply to substitute the new identifier for the old. Figure 5 shows the final product. Note that the package variables `Political_Red_Color`, `Political_Blue_Color`, etc. will also require this conversion in their declarations if the assignment is to compile properly.

The converted code is shown in Figure 5.

4.1.3 Advantages

This type of approach works well with small applications having limited X functionality. It offers the following advantages:

- *Little knowledge of X required.* Since the method is somewhat algorithmic, a programmer with only a basic knowledge of X can do the simple comparisons needed.

- *No redesign required.* In essence, this is a line-by-line translation and avoids the need for extensive redesign of the application to fit a new binding.

- *Fast turn-around times.* Because it is a translation process, and because no design is required, the conversion can proceed rapidly.

4.1.4 Disadvantages

While tempting to use, this method fails on larger applications. As Moore [4] discovered, line-by-line translation quickly becomes unmanageable as more functions are introduced that interact with each other. In general, the disadvantages are categorized as follows:

- *Not robust.* A line-by-line translation is only effective if the number of translations are small, and they match up well with the target bindings. For example, this method fails when a STARS target function such as `Xt_Set_Widget` is encountered since there is no corresponding Xt identifier.

- *No optimization.* Since the STARS bindings were developed, X has added new functions that can significantly reduce the amount of code required. Since low-level mapping does not allow for redesign, these new features will not be used.

- *Prone to error.* Larger applications may have X values that are global in scope (such as default drawables); indiscriminate conversions of these values may introduce side-effects. A STARS binding may have a base type that maps to a similar, but incorrect target binding identifier. For example, `XT.Arg_List` in STARS appears to map to `Xt.Xt_Ancillary_Types.Arg_List_Ptr` in SERC (both are access types). If this mapping is used, the corresponding SERC functions using an argument list will not compile—they require `Xt.Xt_Ancillary_Types.Xt_Arg_List`.

4.2 High-Level Mapping Strategy

In larger applications, a more effective method of conversion will be high-level mapping. Using this approach the programmer takes the STARS application and breaks it down into sections of code that execute specific X tasks. These sections of code are then converted to the new set of bindings.

4.2.1 Method

1. *Identify Sections of X Functionality.* Starting from the X application startup procedures, identify areas where specific tasks are being accomplished. For instance, an application may have a task that creates a form. This may have subtasks that create pushbutton and label widgets.

2. *Develop an Equivalent Set of X Calls.* For each section identified, develop an equivalent set of X calls. Care should be taken that the set of calls be extracted from a version of X that the new bindings are compatible with. The important idea here is to create the same functionality for the application using the best available X functions, procedures, and types. This may require redesigning some areas of the code.

3. *Map Equivalent Set to Target Bindings.* The next task is to map the X calls to Ada bindings. Commercially supported bindings such as Ada/Motif generally have supporting documentation that assists with this task.

4. *Convert Global Variables.* Care must taken that X global values which are used by different sections of code get converted. This will tend to happen naturally: as sections of code are converted, the global variables will become apparent.

These steps should be used as guidelines, and not be interpreted as strict procedure. Conversion from thin to thick bindings, for instance, may allow a relaxation on the equivalent set of X calls since there may not be a direct mapping.

4.2.2 Example Conversion

In this example, another section of Saber will be converted. This piece of code, shown in Figure 6, creates a label on the system's startup form.

1. *Identify Sections of X Functionality.* The code in Figure 6 is a good example of an X task. It creates a label with a string in it. There is no need for additional breakdown of the code.

2. *Develop Equivalent Set of X Calls.* An equivalent set of X calls to perform this function are:

```
XmStringCreateLtoR(text, charset)
XtSetArg(arg, resource_name, value)
XtCreateManagedWidget(name,
                      widget_class,
                      parent,
                      args,
                      num_args)
XmStringFree(string)
```

   Note that X provides a function (`XtVaCreateManagedWidget`) that eliminates the need for an argument list. Ada, however, cannot use this function since it has a variable argument parameter list. The Ada/Motif binding maps this function to `XtCreateManagedWidget` instead.

3. *Map Equivalent Set to Target Bindings.* Each function is mapped to an equivalent set of Ada/Motif bindings. Note that because Ada can constrain an array, the `num_args` parameter on `XtCreateManagedWidget` is not used.

4. *Convert Global Variables.* As the mapping takes place, the variables `Args` and `Form_Widget` will have to be converted. Rather than try a low-level mapping conversion, it is better to determine what `Xt_Create_Managed_Widget` expects them to be. Change them to the appropriate type, in this case `Xt_Arg_List` and `Widget` respectively.

The results of the conversion are shown in Figure 7.

4.2.3 Advantages

This method works well on applications of any size, and has several advantages over low-level mapping:

- *Robust.* Since the strategy develops a set of X calls and maps these to the new bindings, it doesn't fail when a non-mappable STARS function or identifier is found. In Figure 6, `Xt_Set_Widget` will not appear in the equivalent set or the final code.

- *Enhanced performance.* Some new functions in the X Window System (such as `CreateManagedWidget`) take the place of several older calls. Using these in Ada eliminates extra bindings and makes the code execute more efficiently.

```
.
.
.
                    -- Create the TITLE
Label_Text := XM.Xm_String_Create_L_To_R(
                 "Welcome to the SABER Post-Processing System",
                 XM.Xm_STRING_DEFAULT_CHARSET );
XT.Xt_Make_Arg_List( SIZE => 3, ARGS => Args );
XT.Xt_Set_Arg( Args, XM.XmN_Width, 4000 );
XT.Xt_Set_Arg( Args, XM.XmN_Unit_Type, XM.Xm_1000TH_INCHES );
XT.Xt_Set_Arg( Args, XM.XmN_Label_String, Label_Text );
Title_Label := XM.Xm_Create_Label( Form_Widget, "Title", Args );
XT.Xt_Set_Widget( Child_List, Title_Label );
XT.Xt_Clear_Arg_List( Args );
XM.Xm_String_Free( Label_Text );
.
.
.
```

Figure 6: Section of Saber's STARS code

```
.
.
.
                 -- Create the TITLE
Label_Text := Xm_String_Create_L_To_R(
                 "Welcome to the SABER Post-Processing System",
                  Xm_String_Default_Charset );
Xt_Set_Arg( Args(1), Xm_N_Width, 4000 );
Xt_Set_Arg( Args(2), Xm_N_unit_Type, Xm1000th_Inches );
Xt_Set_Arg( Args(3), Xm_N_Label_String, Label_Text );
Title_Label := Xt_Create_Managed_Widget("Title",
                                  Xm_Label_Widget_Class,
                                  Form_Widget,
                                  Args(1 .. 3));
Xm_String_Free( Label_Text );
.
.
.
```

Figure 7: High-level conversion

- *Avoids type incompatibilities.* One of the problems in low-level mapping that high-level mapping avoids is mismatching argument types. Where it was possible to think `Xt.Xt_Ancillary_Types.Arg_List_Ptr` should be used where `XT.Arg_List` had appeared using low-level mapping, it would be immediately obvious that `Xt.Xt_Ancillary_Types.Xt_Arg_List` is used as a parameter type for all Ada/Motif set argument list functions.

4.2.4 Disadvantages

High-level mapping is inherently more complex than low-level mapping. While it handles many cases that low-level can't, there are tradeoffs:

- *Requires expertise in X.* The process for doing this type of conversion is not strictly defined. Programmers will need experience with X to identify functional areas and the best set of equivalent calls to use.

- *Slower turn-around times.* Using the newer features may require design changes in many areas of the application. Changing the design, creating equivalent calls, and mapping them to bindings will require significantly more time than line-by-line translation.

5 Conclusion

This paper discussed two strategies for converting from the STARS binding set to newer commercial bindings. Low-level mapping is appropriate for small applications with limited X functionality, but does not work well with large applications. High-level mapping works well with all applications and produces cleaner code, but requires expertise with X programming.

The high-level mapping strategy was developed after determining that Saber could not be effectively converted using low-level mapping. This method has proved to be effective, allowing us [illegible] convert the initialization, main controller, and game play utilities (about 2000 lines of Ada code) in one week's worth of work. It was found that certain X tasks are repeated often, and conversion techniques can duplicated for each. For example, Figure 6, which creates a label, is representative of about 50% of all the Saber code. Comparison to other examples [2, 5, 6, 7] verifies that this is characteristic of most X programs.

As it tends to produce a better product, our recommendation is to use the high-level mapping strategy for all but very simple systems.

Acknowledgements

The research for this paper was supported by a grant from the Air Force Wargaming Center, AU CADRE/WG, Maxwell AFB, AL, 36112.

References

[1] Freese, Tim. "Ada/X Interface: Binding versus Implementation." *Tri-Ada 1992, Conference Proceedings*. 478. 1992.

[2] Heller, Dan. *Motif Programming Manual*. O'Reilly & Associates, Inc., 1991.

[3] Klabunde, Gary Wayne. *An Animated Graphical Postprocessor for the Saber Wargame*. MS thesis, Air Force Institute of Technology, 1991.

[4] Moore, Donald Ray. *An Enhanced User Interface for the Saber Wargame*. MS thesis, Air Force Institute of Technology, 1992.

[5] Nye, Adrian. *Xlib Programming Manual*. O'Reilly & Associates, Inc., 1990.

[6] Nye, Adrian and Tim O'Reilly. *X Toolkit Intrinsics Programming Manual*. O'Reilly & Associates, Inc., 1990.

[7] Systems Engineering Research Corporation. *Ada / Motif User's Manual*, 1992.

[8] Young, Douglas A. *X Window Systems Programming and Applications with Xt*. Prentice Hall, Inc., 1989.

REPORT DOCUMENTATION PAGE	Form Approved OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 18 March 1993	3. REPORT TYPE AND DATES COVERED Technical Report
4. TITLE AND SUBTITLE Ada/X Window System Bindings: Conversion Strategies		5. FUNDING NUMBERS
6. AUTHOR(S) Karl S. Mathias, Capt, USAF; Mark A. Roth, Maj, USAF		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-7765		8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/EN-TR-93-2
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Wargaming Center, AU CADRE/WG, Maxwell AFB AL, 36112-5532		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution Unlimited		12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The X Window System has come to be accepted as the standard for developing graphical user interfaces on mid to high range workstations. Ada interfaces to X through the use of bindings were developed in the late 1980's under the Software Technology for Adaptable Reliable Systems (STARS) contracts. The bindings have not been maintained, and as they become more obsolete, many are converting their applications to newer sets of commercially available bindings. This paper discusses how the bindings work and two strategies for converting out of the STARS bindings.

14. SUBJECT TERMS Ada, X Windows, Motif, Ada Bindings			15. NUMBER OF PAGES 11
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL